

## 15 | 基础篇：Linux内存是怎么工作的？

2018-12-24 倪朋飞



### 15 | 基础篇：Linux内存是怎么工作的？

朗读人：冯永吉 15'19" | 14.04M

你好，我是倪朋飞。

前几节我们一起学习了 CPU 的性能原理和优化方法，接下来，我们将进入另一个板块——内存。

同 CPU 管理一样，内存管理也是操作系统最核心的功能之一。内存主要用来存储系统和应用程序的指令、数据、缓存等。

那么，Linux 到底是怎么管理内存的呢？今天，我就来带你一起来看看这个问题。

### 内存映射

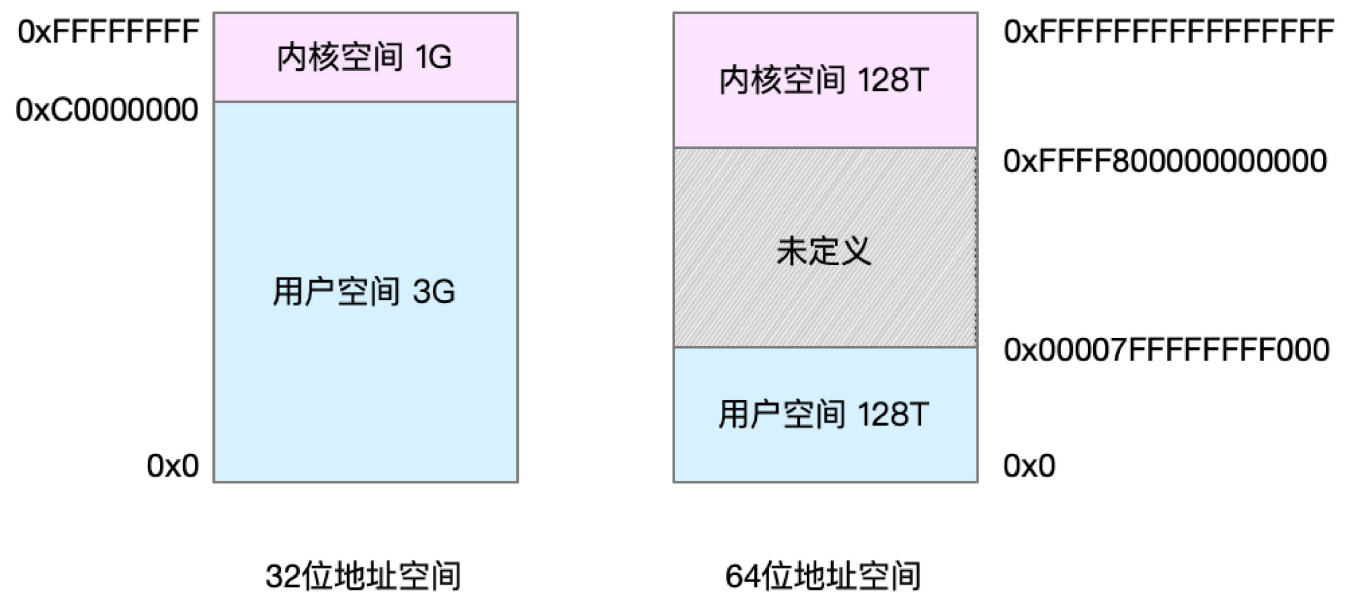
说到内存，你能说出你现在用的这台计算机内存有多大吗？我估计你记得很清楚，因为这是我们购买时，首先考虑的一个重要参数，比方说，我的笔记本电脑内存就是 8GB 的。

我们通常所说的内存容量，就像我刚刚提到的 8GB，其实指的是物理内存。物理内存也称为主存，大多数计算机用的主存都是动态随机访问内存（DRAM）。只有内核才可以直接访问物理

内存。那么，进程要访问内存时，该怎么办呢？

Linux 内核给每个进程都提供了一个独立的虚拟地址空间，并且这个地址空间是连续的。这样，进程就可以很方便地访问内存，更确切地说是访问虚拟内存。

虚拟地址空间的内部又被分为**内核空间**和**用户空间**两部分，不同字长（也就是单个 CPU 指令可以处理数据的最大长度）的处理器，地址空间的范围也不同。比如最常见的 32 位和 64 位系统，我画了两张图来分别表示它们的虚拟地址空间，如下所示：

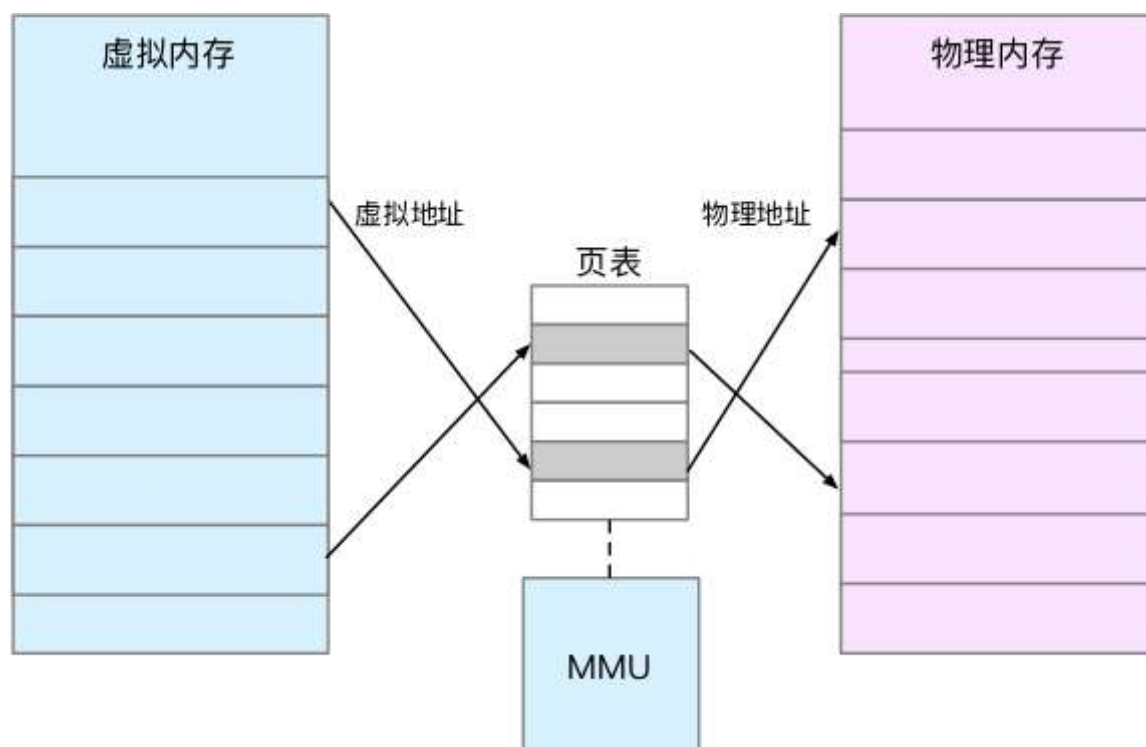


通过这里可以看出，32 位系统的内核空间占用 1G，位于最高处，剩下的 3G 是用户空间。而 64 位系统的内核空间和用户空间都是 128T，分别占据整个内存空间的最高和最低处，剩下的中间部分是未定义的。

还记得进程的用户态和内核态吗？进程在用户态时，只能访问用户空间内存；只有进入内核态后，才可以访问内核空间内存。虽然每个进程的地址空间都包含了内核空间，但这些内核空间，其实关联的都是相同的物理内存。这样，进程切换到内核态后，就可以很方便地访问内核空间内存。

既然每个进程都有一个这么大的地址空间，那么所有进程的虚拟内存加起来，自然要比实际的物理内存大得多。所以，并不是所有的虚拟内存都会分配物理内存，只有那些实际使用的虚拟内存才分配物理内存，并且分配后的物理内存，是通过**内存映射**来管理的。

内存映射，其实就是将**虚拟内存地址**映射到**物理内存地址**。为了完成内存映射，内核为每个进程都维护了一张页表，记录虚拟地址与物理地址的映射关系，如下图所示：



页表实际上存储在 CPU 的内存管理单元 MMU 中，这样，正常情况下，处理器就可以直接通过硬件，找出要访问的内存。

而当进程访问的虚拟地址在页表中查不到时，系统会产生一个**缺页异常**，进入内核空间分配物理内存、更新进程页表，最后再返回用户空间，恢复进程的运行。

另外，我在 [CPU 上下文切换的文章中](#)曾经提到，TLB（Translation Lookaside Buffer，转译后备缓冲器）会影响 CPU 的内存访问性能，在这里其实就可以得到解释。

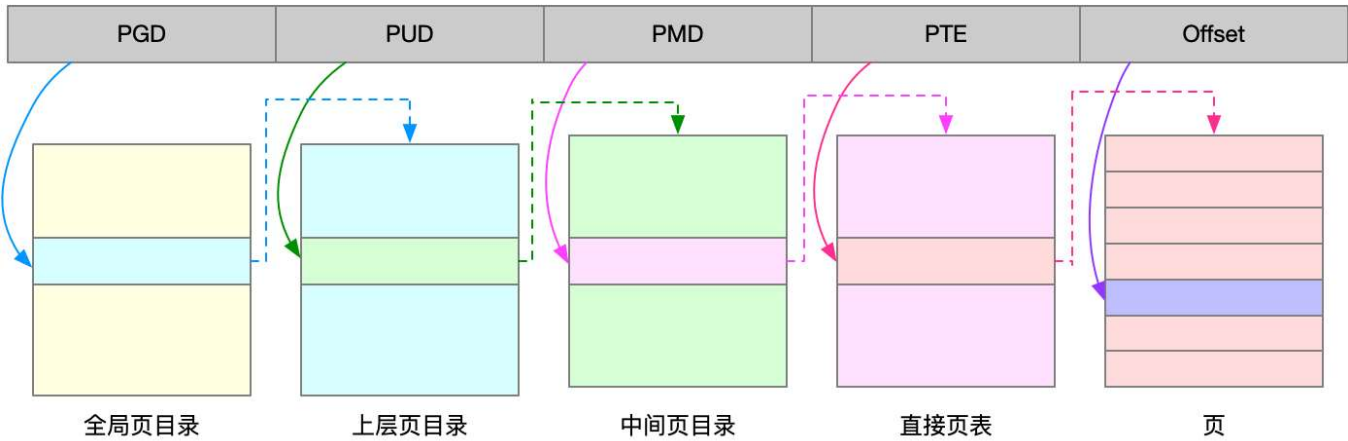
TLB 其实就是 MMU 中页表的高速缓存。由于进程的虚拟地址空间是独立的，而 TLB 的访问速度又比 MMU 快得多，所以，通过减少进程的上下文切换，减少 TLB 的刷新次数，就可以提高 TLB 缓存的使用率，进而提高 CPU 的内存访问性能。

不过要注意，MMU 并不以字节为单位来管理内存，而是规定了一个内存映射的最小单位，也就是页，通常是 4 KB 大小。这样，每一次内存映射，都需要关联 4 KB 或者 4KB 整数倍的内存空间。

页的大小只有 4 KB，导致的另一个问题就是，整个页表会变得非常大。比方说，仅 32 位系统就需要 100 多万个页表项（4GB/4KB），才可以实现整个地址空间的映射。为了解决页表项过多的问题，Linux 提供了两种机制，也就是多级页表和大页（HugePage）。

多级页表就是把内存分成区块来管理，将原来的映射关系改成区块索引和区块内的偏移。由于虚拟内存空间通常只用了很少一部分，那么，多级页表就只保存这些使用中的区块，这样就可以大大地减少页表的项数。

Linux 用的正是四级页表来管理内存页，如下图所示，虚拟地址被分为 5 个部分，前 4 个表项用于选择页，而最后一个索引表示页内偏移。

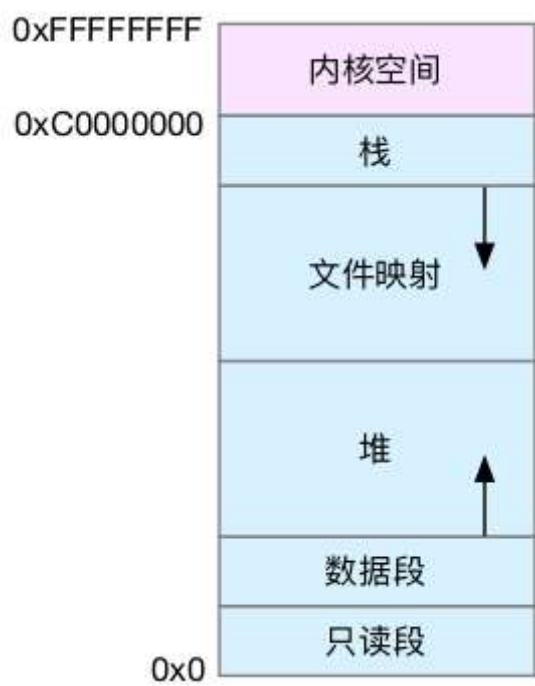


再看大页，顾名思义，就是比普通页更大的内存块，常见的大小有 2MB 和 1GB。大页通常用在使用大量内存的进程上，比如 Oracle、DPDK 等。

通过这些机制，在页表的映射下，进程就可以通过虚拟地址来访问物理内存了。那么具体到一个 Linux 进程中，这些内存又是怎么使用的呢？

### 虚拟内存空间分布

首先，我们需要进一步了解虚拟内存空间的分布情况。最上方的内核空间不用多讲，下方的用户空间内存，其实又被分成了多个不同的段。以 32 位系统为例，我画了一张图来表示它们的关系。



通过这张图你可以看到，用户空间内存，从低到高分别是五种不同的内存段。

1. 只读段，包括代码和常量等。

2. 数据段，包括全局变量等。
3. 堆，包括动态分配的内存，从低地址开始向上增长。
4. 文件映射段，包括动态库、共享内存等，从高地址开始向下增长。
5. 栈，包括局部变量和函数调用的上下文等。栈的大小是固定的，一般是 8 MB。

在这五个内存段中，堆和文件映射段的内存是动态分配的。比如说，使用 C 标准库的 `malloc()` 或者 `mmap()`，就可以分别在堆和文件映射段动态分配内存。

其实 64 位系统的内存分布也类似，只不过内存空间要大得多。那么，更重要的问题来了，内存究竟是怎么分配的呢？

## 内存分配与回收

`malloc()` 是 C 标准库提供的内存分配函数，对应到系统调用上，有两种实现方式，即 `brk()` 和 `mmap()`。

对小块内存（小于 128K），C 标准库使用 `brk()` 来分配，也就是通过移动堆顶的位置来分配内存。这些内存释放后并不会立刻归还系统，而是被缓存起来，这样就可以重复使用。

而大块内存（大于 128K），则直接使用内存映射 `mmap()` 来分配，也就是在文件映射段找一块空闲内存分配出去。

这两种方式，自然各有优缺点。

`brk()` 方式的缓存，可以减少缺页异常的发生，提高内存访问效率。不过，由于这些内存没有归还系统，在内存工作繁忙时，频繁的内存分配和释放会造成内存碎片。

而 `mmap()` 方式分配的内存，会在释放时直接归还系统，所以每次 `mmap` 都会发生缺页异常。在内存工作繁忙时，频繁的内存分配会导致大量的缺页异常，使内核的管理负担增大。这也是 `malloc` 只对大块内存使用 `mmap` 的原因。

了解这两种调用方式后，我们还需要清楚一点，那就是，当这两种调用发生后，其实并没有真正分配内存。这些内存，都只在首次访问时才分配，也就是通过缺页异常进入内核中，再由内核来分配内存。

整体来说，Linux 使用伙伴系统来管理内存分配。前面我们提到过，这些内存存在 MMU 中以页为单位进行管理，伙伴系统也一样，以页为单位来管理内存，并且会通过相邻页的合并，减少内存碎片化（比如 `brk` 方式造成的内存碎片）。

你可能会想到一个问题，如果遇到比页更小的对象，比如不到 1K 的时候，该怎么分配内存呢？

实际系统运行中，确实有大量比页还小的对象，如果为它们也分配单独的页，那就太浪费内存了。

所以，在用户空间，`malloc` 通过 `brk()` 分配的内存，在释放时并不立即归还系统，而是缓存起来重复利用。在内核空间，Linux 则通过 `slab` 分配器来管理小内存。你可以把 `slab` 看成构建在伙伴系统上的一个缓存，主要作用就是分配并释放内核中的小对象。

对内存来说，如果只分配而不释放，就会造成内存泄漏，甚至会耗尽系统内存。所以，在应用程序用完内存后，还需要调用 `free()` 或 `unmap()`，来释放这些不用的内存。

当然，系统也不会任由某个进程用完所有内存。在发现内存紧张时，系统就会通过一系列机制来回收内存，比如下面这三种方式：

- 回收缓存，比如使用 LRU (Least Recently Used) 算法，回收最近使用最少的内存页面；
- 回收不常访问的内存，把不常用的内存通过交换分区直接写到磁盘中；
- 杀死进程，内存紧张时系统还会通过 OOM (Out of Memory)，直接杀掉占用大量内存的进程。

其中，第二种方式回收不常访问的内存时，会用到交换分区（以下简称 Swap）。Swap 其实就是把一块磁盘空间当成内存来用。它可以把进程暂时不用的数据存储在磁盘中（这个过程称为换出），当进程访问这些内存时，再从磁盘读取这些数据到内存中（这个过程称为换入）。

所以，你可以发现，Swap 把系统的可用内存变大了。不过要注意，通常只在内存不足时，才会发生 Swap 交换。并且由于磁盘读写的速度远比内存慢，Swap 会导致严重的内存性能问题。

第三种方式提到的 OOM (Out of Memory)，其实是内核的一种保护机制。它监控进程的内存使用情况，并且使用 `oom_score` 为每个进程的内存使用情况进行评分：

- 一个进程消耗的内存越大，`oom_score` 就越大；
- 一个进程运行占用的 CPU 越多，`oom_score` 就越小。


这样，进程的 `oom_score` 越大，代表消耗的内存越多，也就越容易被 OOM 杀死，从而可以更好保护系统。

当然，为了实际工作的需要，管理员可以通过 `/proc` 文件系统，手动设置进程的 `oom_adj`，从而调整进程的 `oom_score`。

`oom_adj` 的范围是 `[-17, 15]`，数值越大，表示进程越容易被 OOM 杀死；数值越小，表示进程越不容易被 OOM 杀死，其中 `-17` 表示禁止 OOM。

比如用下面的命令，你就可以把 sshd 进程的 oom\_adj 调小为 -16，这样，sshd 进程就不容易被 OOM 杀死。

```
1 echo -16 > /proc/$(pidof sshd)/oom_adj
```

 复制代码

## 如何查看内存使用情况

通过了解内存空间的分布，以及内存的分配和回收，我想你对内存的工作原理应该有了大概的认识。当然，系统的实际工作原理更加复杂，也会涉及其他一些机制，这里我只讲了最主要的原理。掌握了这些，你可以对内存的运作有一条主线认识，不至于脑海里只有术语名词的堆砌。


那么在了解内存的工作原理之后，我们又该怎么查看系统内存使用情况呢？

其实前面 CPU 内容的学习中，我们也提到过一些相关工具。在这里，你第一个想到的应该是 free 工具吧。下面是一个 free 的输出示例：

```
1 # 注意不同版本的 free 输出可能会有所不同
```

```
2 $ free
```

	total	used	free	shared	buff/cache	available
Mem:	8169348	263524	6875352	668	1030472	7611064
Swap:	0	0	0			

 复制代码


你可以看到，free 输出的是一个表格，其中的数值都默认以字节为单位。表格总共有两行六列，这两行分别是物理内存 Mem 和交换分区 Swap 的使用情况，而六列中，每列数据的含义分别为：

- 第一列，total 是总内存大小；
- 第二列，used 是已使用内存的大小，包含了共享内存；
- 第三列，free 是未使用内存的大小；
- 第四列，shared 是共享内存的大小；
- 第五列，buff/cache 是缓存和缓冲区的大小；
- 最后一列，available 是新进程可用内存的大小。

这里尤其注意一下，最后一列的可用内存 available。available 不仅包含未使用内存，还包括了可回收的缓存，所以一般会比未使用内存更大。不过，并不是所有缓存都可以回收，因为有些缓存可能正在使用中。



不过，我们知道，free 显示的是整个系统的内存使用情况。如果你想查看进程的内存使用情况，可以用 top 或者 ps 等工具。比如，下面是 top 的输出示例：

 复制代码

```
1 # 按下 M 切换到内存排序
2 $ top
3 ...
4 KiB Mem : 8169348 total, 6871440 free, 267096 used, 1030812 buff/cache
5 KiB Swap: 0 total, 0 free, 0 used. 7607492 avail Mem
6
7
8 PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
9 430 root 19 -1 122360 35588 23748 S 0.0 0.4 0:32.17 systemd-journal
10 1075 root 20 0 771860 22744 11368 S 0.0 0.3 0:38.89 snapd
11 1048 root 20 0 170904 17292 9488 S 0.0 0.2 0:00.24 networkd-dispat
12 1 root 20 0 78020 9156 6644 S 0.0 0.1 0:22.92 systemd
13 12376 azure 20 0 76632 7456 6420 S 0.0 0.1 0:00.01 systemd
14 12374 root 20 0 107984 7312 6304 S 0.0 0.1 0:00.00 sshd
15 ...
```

top 输出界面的顶端，也显示了系统整体的内存使用情况，这些数据跟 free 类似，我就不再重复解释。我们接着看下面的内容，跟内存相关的几列数据，比如 VIRT、RES、SHR 以及 %MEM 等。

这些数据，包含了进程最重要的几个内存使用情况，我们挨个来看。

- VIRT 是进程虚拟内存的大小，只要是进程申请过的内存，即便还没有真正分配物理内存，也会计算在内。
- RES 是常驻内存的大小，也就是进程实际使用的物理内存大小，但不包括 Swap 和共享内存。
- SHR 是共享内存的大小，比如与其他进程共同使用的共享内存、加载的动态链接库以及程序的代码段等。
- %MEM 是进程使用物理内存占系统总内存的百分比。

除了要认识这些基本信息，在查看 top 输出时，你还要注意两点。

第一，虚拟内存通常并不会全部分配物理内存。从上面的输出，你可以发现每个进程的虚拟内存都比常驻内存大得多。

第二，共享内存 SHR 并不一定是共享的，比方说，程序的代码段、非共享的动态链接库，也都算在 SHR 里。当然，SHR 也包括了进程间真正共享的内存。所以在计算多个进程的内存使用时，不要把所有进程的 SHR 直接相加得出结果。



## 小结

今天，我们梳理了 Linux 内存的工作原理。对普通进程来说，它能看到的其实是内核提供的虚拟内存，这些虚拟内存还需要通过页表，由系统映射为物理内存。

当进程通过 `malloc()` 申请内存后，内存并不会立即分配，而是在首次访问时，才通过缺页异常陷入内核中分配内存。

由于进程的虚拟地址空间比物理内存大很多，Linux 还提供了一系列的机制，应对内存不足的问题，比如缓存的回收、交换分区 Swap 以及 OOM 等。

当你需要了解系统或者进程的内存使用情况时，可以用 `free` 和 `top`、`ps` 等性能工具。它们都是分析性能问题时最常用的性能工具，希望你能熟练使用它们，并真正理解各个指标的含义。

## 思考

最后，我想请你来聊聊你所理解的 Linux 内存。你碰到过哪些内存相关的性能瓶颈？你又是怎么样来分析它们的呢？你可以结合今天学到的内存知识和工作原理，提出自己的观点。

欢迎在留言区和我讨论，也欢迎你把这篇文章分享给你的同事、朋友。我们一起在实战中演练，在交流中进步。



# Linux 性能优化实战

10 分钟帮你找到系统瓶颈

倪朋飞

微软资深工程师  
Kubernetes 项目维护者



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

©版权归极客邦科技所有，未经许可不得转载

精选留言



我来也

👍 14

[D15打卡]

linux的内存跟windows的很不一样。类linux的系统会尽量使用内存缓存东西，提供运行效率。所以linux/mac显示的free剩余内存通常很小，但实际上被缓存的cache可能很大，并不代表系统内存紧张！

曾经就闹过笑话，看见系统free值很低，怕程序因为oom被系统杀掉，还特意写个c程序去挤内存。程序不停的申请1MB内存然后memset，随机挑几个位置写，保证申请的都被加载到物理内存中。（跟文中描述的一致，只申请不使用不会加载到物理内存）然后挤的差不多了就把测试程序关掉。看上去free变大了很多很开心。现在想想，就是掩耳盗铃罢了。

以前物理机上还有swap交换分区，现在都是云服务器，基本没有了该分区。也不会遇到因为频繁使用交换分区导致性能下降的问题了。

我内存方面的问题遇到的都比较简单，基本上就是top/free看看系统和各程序的，找到有问题的程序，看看是否有内存泄露。平常不泄漏都是够用的。

redis对内存比较敏感，曾经就因为配置项是默认值，在内存用完后，所有的set操作都直接返回错误，导致线上系统故障。（redis在备份时会新开一个进程，实际使用内存量会翻番。）

后来会定期检查redis的info memory看内存使用情况。

---

期待的内存篇开始了，好开心！又可以跟着老师学新知识啦！

2018-12-24

作者回复

👍 谢谢分享你的经历。缓存和swap后面都还会细讲

2018-12-25



JohnT3e

👍 10

目前我配合使用《Operating System Concepts》这本书学习此专栏，也是一个不错的选择。此书中的第四部分“Memory Management”对今天这部分内容有较为详细的描述，感兴趣的同学可以去看一下。另外对于TOP命令中输出的内存情况解释，可以认真看一下man手册的内容。如果你动手能力比较强，可以看<https://blog.holbertonschool.com/hack-the-virtual-memory-malloc-the-heap-the-program-break/>这篇博客，手把手教你使用程序（C语言）来实际地去搞清楚虚拟内存分布。

2018-12-24

作者回复

👍 谢谢分享，这篇博客很详细

2018-12-24



espzest

👍 7

用户空间malloc不会使用slab机制吧，slab是内核空间的，对不？

2018-12-24

## 作者回复

嗯嗯，是的，谢谢指出。slab是内核空间的，只用来管理内核中的小块内存

2018-12-24



虎虎♡

👍 5

请教老师几个问题

1. 当内存紧张时，系统通过三种机制回收内存。第二种换页比较好理解，但是第一种LRU回收内存页怎么理解？回收后的页去哪了？如果直接删除会导致程序出问题吗？

2. OOM的分数是参照进程的实际消耗内存还是虚拟内存大小？

3. 进程启动时，是不是需要分配一个最小的内存？都包括什么呢？如何确定最小的内存是多大呢？比如我在k8s里设置container的request值，我希望能容纳下这个container的最小内存，有没有办法计算呢？

有的问题比较小白，望老师包含。如果无法简短的回答，能否推荐些资料呢？谢谢

2018-12-24



赵强强

👍 2

倪老师，“Operating Systems Design and Implementation”里面说，大部分页式存储管理，页表是存储到内存里面的，为降低频繁访问内存计算物理地址，引入TLB用于缓存常用映射以提升性能。以现在流行的linux为例，本节课说页表存储在MMU是否还正确呢？

2018-12-24

## 作者回复

MMU全称就是内存管理单元，管理地址映射关系（也就是页表）。但MMU的性能跟CPU比还是不够快，所以又有了TLB。TLB实际上是MMU的一部分，把页表缓存起来以提升性能。

2018-12-24



赵强强

👍 1

倪老师，如果进程间的虚拟内存空间独立，每个进程又有自己的页表，那么进程怎么获知其他进程占用的物理内存情况，怎么防止覆盖其他进程的物理内存块呢？

2018-12-24

## 作者回复

1. 进程不需要获取其他进程的内存情况

2. 所有进程的内存都是由内核来管理的，内核保证内存的访问安全。比如，访问非法地址时，进程会panic

2018-12-24



圣域烈焰

👍 1

请教一个问题，使用free命令的时候，看到的buffer、cache和shared有什么区别？

2018-12-24

| 作者回复

下一期讲 buffer 和 cache

2018-12-25



子轩Zixuan

👍 1

cpu和内存应该是出问题频率最高的两块内容了，期待内存篇实战

2018-12-24

| 作者回复

马上就开始😊

2018-12-24



大甜菜

👍 1

malloc本身不会使用slab

只有内核中使用kmalloc才会通过slab分配内存

2018-12-24

| 作者回复

嗯嗯，是的，谢谢指出

2018-12-24



wahaha

👍 1

请问，当手动让一个进程暂停后，如何手动让内核立即swap out该进程占用的内存？

2018-12-24



wahaha

👍 1

老师，请问怎么监控和分析系统的swap指标？如果用了zswap或swap on zram device，能否得到它的压缩和解压的CPU使用情况？

2018-12-24



Linuxer

👍 1

有问题请教，上面提到的段最终是否都是按页映射，段的大小有限制吗？

2018-12-24



Linuxer

👍 1

我所碰到的内存问题有：频繁的缺页中断导致的，内存不足导致的，malloc锁争用导致的，64位系统物理内存还有几十G但是分配内存失败。不知道这些课程会不会都有涉及？

2018-12-24

| 作者回复

前两个有涉及，后面两个问题有没有细节？我可以放在答疑中解答

2018-12-25



zylv

👍 0

老师，后面会讲cgroup吗，我遇到一个内存使用的问题，cgroup下的memory\_in\_usage\_bytes跟free中used 内存差几倍，这是为啥

2018-12-29



Geek\_37593b

👍 0

先打卡，地铁上。

2018-12-28



React

👍 0

老师好，phpfpm可以分配多个子进程，假设主机中只运行phpfpm.我在计算phpfpm能够分配最大进程数时需要知道单个phpfpm进程占用的内存大小，这个占用的内存大小，我只需要计算实际占用的物理内存，即top中的res即可吗？另外通过这篇文章，我知道了为什么系统中进程占用的virt会比res和shr的总和还要大了。

2018-12-28

作者回复

实际上内存的占用需要做压力测试才能知道，因为繁忙进程一般会比空闲时占用更多的内存。在计算进程实际内存的时候，用PSS更准确。

2018-12-28



Only now  
mark

👍 0

2018-12-27



Griffin

👍 0

hmmm，我要提问。老师，我的开发机是macbook。内存是16G。但是经常出现已12G，swap用了3-4G。我平时没用ide就tmux+vim，内存大户就一个dorker和chrome。卡的我不得了，每次都只能重启来缓解。发现有个windowserver占用很多。感觉16G内存开发应该完全够用才对啊，有没有mac下可以禁用swap？让内存不够的时候优先回收缓存和不常用的内存，而不是滥用swap？

2018-12-27

作者回复

MAC还真不了解。接下来还有swap的文章，到时候可以参考试试

2018-12-28



魏峰

👍 0

Linux没有段机制？我看有的文档上说有

2018-12-27

作者回复

有的，篇幅限制，文中没有展开

2018-12-28



13001236383

👍 0

老师请教两个问题第1.每个用户进程在运行时都会为用户空间的各段空间内申请自己的空间，比如说每个进程都会在栈空间申请自己的栈，在堆空间申请自己的堆空间？各进程是怎么实现访问隔离的？

2.用户态内存和内核态内存是不是只从用户进程角度来看？内核使用内存是直接分配了，还是也是需要进行映射？多谢老师。

2018-12-27

#### 作者回复

1. 内核保证内存的隔离

2. 内核中大部分是直接分配物理页，但是也可以通过vmalloc来分配不连续的物理内存，这时候必须通过页表才能知道映射关系

2018-12-28